# Java CoG Kit Reference Manual (Draft)

Gregor von Laszewski, Jarek Gawor, Warren Smith, Peter Lane, Steve Tuecke, Ian Foster

October 18, 1999

Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, U.S.A.
http://www-unix.mcs.anl.gov/~laszewsk/cog
gregor@mcs.anl.gov
Version: 1.1.0

**Abstract**

This document describes the classes and methods of the Java CoG Kit. It is intended as a reference manual rather than a users manual. Nevertheless, we have augmented the reference manual with extensive examples, which allow the reader to see selected usage of important packages, classes, and methods.

## Contents

## 1 Terminology

In Java the term “interface” has a specific meaning. Thus, to prevent inconsistencies and confusion, we decided not to use the term “application interface”, commonly used with procedural programming languages. Together, packages, classes, interfaces, and methods describe the details of the Java CoG kit sufficiently.

## 2 Java CoG-Kit

The Java CoG-Kit provides a convenient mapping between a selected set of Grid toolkit components and the Java programming languages, and its concepts are described in [**?**]. The Java classes use the object-oriented features of Java and do not necessarily transform into a one-to-one mapping between functions defined in the Grid toolkit. The current version is based on Globus 1.0 but the port to 1.1 is under development.

## 3 Contribution

To allow the concurrent development of the Java CoG-Kit as acommunity effort, we identified the following coding rules:[1]

**Coding Rules:**

1. All user-accessible classes must be formulated as JavaBeans.

2. All classes must be documented with JavaDoc.

3. All classes should be augmented and demonstrated with easy to follow examples.

## 4 Availability

**Alpha Release.** Currently, the code can only obtained with prior approval from Gregor von Laszewski and Ian Foster; send mail to gregor@mcs.anl.gov. We would like to keep the distribution small because the code it is an early alpha release.
The current alpha release is maintained in a CVS directory. The location is:

> <on purpose not mentioned here, but will be filled in later>

To participate in the alpha development, one needs to have an account on pitcairn.mcs.anl.gov and access the CVS directory through ssl. In addition, a recent snapshot of the alpha release can be retrieved from

> http://www-unix.mcs.anl.gov/~laszewsk/cog/<on purpose not mentioned here, but will be filled in later>

**Beta Release.** The first beta release is expected at SC’99. The version number will be version 1.1.0

**Official and Current Release.** No official release is yet available. With the release of a new version, prior versions will no longer be supported by the developers.

[1] At present not all coding rules are enforecd, but with the release of the version 3.0 we hope to accomplish this.

## 5 Packages

All current classes of the Java CoG Kit are currently part of the package `org.globus.`[2] We distinguish the following major packages, which we will describe in more detail in this document.

- ▷ org.globus.common
- ▷ org.globus.rsl
- ▷ org.globus.gram
- ▷ org.globus.duroc
- ▷ org.globus.mds
- ▷ org.globus.gass
- ▷ org.globus.util
- ▷ org.globus.gui
- ▷ org.globus.hbm
- ▷ org.globus.examples
- ▷ org.globus.gara

# Part I
# Low-Level Packages

### 5.1 Package org.globus.common

The package *common* contains commonly used classes that allow Java CoG-Kit developers and users to access, for example, data structures that are used to define a mapping between the Globus specification language and graphical user interfaces designed in Java.[3]

**Inheritance:**

```
Hashtable
    |
    +
    MVHashtable
         |
         +--- ConstrainedMVHashtable
         |
         +--- RSLAttributes
```

#### 5.1.1 Class MVHashtable

**CLASSNAME:**

*org.globus.common.MVHashtable*

[2]Note: We have to discuss if this should be changed to org.cog and have an org.cog.globus and org.cog.computingportals, ... .
[3]The class MVHashtable is in the current alpha release called GlobusAttributes.

**PURPOSE:**

A MVHashtable defines a *multivalued hashtable;* that is, each entry identified by a key can have multiple values that are stored in a vector as Java objects. We provide methods for access and manipulation similar to those as in the default hastable available in Java.[4]

**METHODS:**

**Initialization**

▷ *MVHashtable()*

▷ *MVHashtable(MVHashtable MVHashtable)*

**Adding Attributes**

For all adding functions the following rule applies: If there is already an attribute with this name, a new value is appended to a vector that is attached with each key.

▷ *void add(MVHashtable multivaluedHashtable)*, adds the attributes specified in multivaluedHashtable .

▷ *void add(String key, Object value)*, adds a value at the end of the vector identified by the key.

▷ *void add(String key, Object[] values)*, adds multiple values at the end of the vector identified by the key.

▷ *void add(String key, Vector values)*, adds multiple values at the end of the vector identified by the key by simply concatenating all specified values.

**Setting an Attribute**

▷ *void set(String key, Object value)*, sets an attribute with the specified key to a single value. Previous values will be removed.

**Removing Attributes**

▷ *void clear()*, deletes all values and keys.

▷ *void remove(String key, int index)*, removes the i-th value that is stored in the vector identified by the key.

▷ *void remove(String key, Object value)*, removes a specific value that is stored in the vector identified by the key.

▷ *void remove(String key)*, all values associated with the key are.

**Testing for Attributes**

▷ *boolean contains(String key, Object value)*, returns true if the value is stored under the specified key.

▷ *boolean containsKey(String key)*, returns true if the key exists.[5]

[4]Note: Do we want to rename it to MultiValuedHashtable, BucketHashtable would also be possible?
[5]alpha release: containsName(...)

**Retrieving Attributes**

▷ *Vector get(String key)*, returns the vector (of objects) associated with the key.

▷ *Object getFirstValue(String key)*, returns the first object associated with the key.

▷ *Object getLastValue(String key)*, returns the last object associated with the key.

▷ *Object getValueAt(String key, int index)*, returns the object at position index associated with the key.

▷ *Enumeration keys()*, returns an enumeration of all keys.[6]

**Debugging**

▷ *void print()*, prints the contents of the data structure.

**Other Query Methods**

▷ *int size()*, returns the number of attributes.

▷ *int size(String name)*, returns the number of values stored in the named attribute.

**Functions not yet implemented**

▷ String toXML() to be defined.[7]

▷ String toPseudoLDIF() to be defined.

▷ String toPseudoLDIF(String colon)

▷ String toString(), see Hashtable.

▷ void rehash(), see Hashtable.

▷ boolean isEmpty(), see Hashtable.

### 5.1.2 Class Constrained Attributes

**CLASSNAME:**

*ConstrainedMVHashtable*
(currently called ConstrainedGlobusAttributes)

**PURPOSE:**

The class *ConstrainedMVHashtable* extends *MVHashtable* and allows one to specify constraints for each key, which determine wether a key is optional, or required, and wether a particular key is single, or multivalued.

**CONSTRUCTORS:**

▷ *ConstrainedMVHashtable()*

▷ *ConstrainedMVHashtable(ConstrainedMVHashtable constrainedMultivaluedHashtable)*

[6] alpha release: names()
[7] The conversion methods could be placed in a seperate conversion class,

**METHODS:**

**Query Methods:**

▷ *boolean isRequired(String key)*, returns true if the key is constrained to be required.

▷ *boolean isSingleValued(String key)*, returns true if the key is constrained to be single valued.

**Set Methods:**

▷ *void setMaximum(String key, int maxValues)*, constrains the number of values for a key.

▷ *void setRequired(String[] keys, boolean required)*, constrains all keys specified in the array of strings to be required.

▷ *void setRequired(String key, boolean required)*, constrains the key to be required.

▷ *void setSingleValued(String[] keys, boolean single)*, constrains all keys specified in the array of strings to be single valued.

▷ *void setSingleValued(String key, boolean single)*, constrains the key to be single valued.

**Debugging:**

▷ *void print()*, prints the data structure in readable format.

### 5.1.3 Class RSLAttributes

**CLASSNAME:**

*RSLAttributes*

**PURPOSE:**

The class *RSLAttributes* extends MVHashtable to function as a data structure that stores Globus RSL information. Thus, graphical user interfaces that access MVHashtable can also be used to manipulate RSL strings.

**METHODS:**

**Initialization**

▷ *RSLAttributes()*

▷ *RSLAttributes(RSLAttributes rslAttributes)*

**Conversion from RSL**

▷ *void fromRSL(String rsl)*, takes the rsl string and initializes the RSLAttributes with its values. Previous values are cleared.

**Conversion to RSL**

▷ *String toRSL()*, returns the RSL as a string

▷ *String toRSL(boolean outsideParens)*, to be documented. Experimental method.

**Other Methods**

- ▷ *void print()*, prints the data structure in readable format.
- ▷ *static String quotify(String str)*, to be documented. Experimental method.

## 5.2 Package org.globus.gram

The package *gram* enables one to easily start Globus/Grid Gram jobs from Java. To support the object oriented features of Java, the object *GramJob* is used to conveniently start and monitor jobs started in the Globus/Grid environment. As result of the definition of a GramJob, we are able to utilize the Java event model and monitor state changes to the GramJob. We distinguish the following classes:

- ▷ org.globus.gram.Gram
  This class is used only for internal purposes and is not called by the user directly.
- ▷ org.globus.gram.GramJob
  This class defines the properties of jobs, submission routines, and cancellation routines.
- ▷ org.globus.gram.GramException
  This class defines the exceptions that are thrown by the various Gram methods.
- ▷ org.globus.gram.GramJobListener
  This class provides a JavaBeans based listener interface to GramJobs .

**Example:** Before we introduce each of the classes and interfaces, we demonstrate the usage with the following example. In this example a job is requested for execution.

```
public class GramExample implements GramJobListener {

  private someMethod() {
    ...

    String gramContact = "pitcairn.mcs.anl.gov:8713:...";
    String rsl         = "&(executable=...)(...)(...)"
    GramJob job = new GramJob(rsl);
    try {
      job.ping(gramContact);
    } catch (GramException e) {
      // can’t submit
      return;
    }

    job.addListener(this)
    try {
      job.request(gramContact);
    } catch (GramException e) {
      // request failed
      ...
    }

    ...

    // the user wants to do something else so...
    try {
      job.cancel();
    } catch (GramException e) {
      // cancel failed
```

```
        ...
      }
    }

    public stateChanged(GramJob job) {
      ...
      // react to a state change
    }
  }
```

### 5.2.1 Class Gram

The class Gram is for internal use. No methods should be called by users.

### 5.2.2 Class GramJob

The class GramJob is the main class for using the Globus GRAM (Globus Resource Allocation Manager) to start jobs on remote computers.

**CLASSNAME:**

*GramJob*

**VARIABLES:**

This class publicly defines static int state variables that represent the possible states of a GramJob.

| State | Description |
|---|---|
| STATE_ACTIVE | see the Globus manual |
| STATE_DONE | - " - |
| STATE_FAILED | - " - |
| STATE_PENDING | - " - |
| STATE_SUSPENDED | - " - |

**METHODS:**

**Initialization**

- ▷ *GramJob(String gramContact, String rsl)*, creates a GramJob with a GRAM contact and a description of the job in RSL format.
- ▷ *GramJob(String rsl)*, creates a GramJob with the RSL.[8]

**Resource Manager Query**

- ▷ *void ping(String gramContact) throws GramException*, tests wether the user can submit a job to a GRAM specified by the contact provided. [9]

[8]Here is an inconsistency in the definition of Globus GRAM. In order to make the RSL similar to Duroc tow attributes should be added to GRAM: label and resourceManagerCon tact. We request to fix this in a later release of Globus.

[9]This function is different form the current implementation. We need to discuss if the suggested add of the gramContact is better.

**Job Handling**

- ▷ *void request(String gramContact) throws GramException*, sends a request to start the GramJob. If an error occurs. This functionality is often named *submit*. However, since we use Globus terminology here the call is called request.[10]
- ▷ *void cancel() throws GramException*, cancels an already requested GramJob.

**Job Query**

- ▷ *String getID()*, returns the ID (GRAM job contact) assigned to the job when it was requested.
- ▷ *int getState()*, returns the state for the GramJob.
- ▷ *String getStateAsString()*, returns a textual description of the GramJob state.
- ▷ *String getGramContact()*, returns the GRAM contact for the GramJob.
- ▷ *String getDescription()*, returns the RSL description of the GramJob. The description depends on which constructor was used.[11]
- ▷ *void addListener(GramJobListener listener)*, adds a listener to the GramJob. All listeners are notified when the state of a GramJob changes.
- ▷ *void removeListener(GramJobListener listener)*, removes a listener from the GramJob. The listener will no longer be notified when the state of the GramJob changes.

### 5.2.3 Class GramException

The class GramException contains many definitions for error codes of the form ERROR_* and also SUCCESS. The error codes are a superset of those defined for the GRAM C client. Additional error codes are added for errors that can occur in the Java code.

**CLASSNAME:**

*GramException*

**VARIABLES:**

The following list of variables is defined. Their specification can be found in the Globus manual:
*BAD_ARGUMENTS,*
*BAD_RSL,*
*BAD_RSL_ENVIRONMENT,*
*DRYRUN,*
*ERROR_ARG_FILE_CREATION_FAILED,*
*ERROR_AUTHORIZATION,*
*ERROR_BAD_DIRECTORY,*
*ERROR_BAD_EXECUTABLE,*
*ERROR_BAD_SCRIPT_ARG_FILE,*
*ERROR_CONNECTION_FAILED,*
*ERROR_CREATING_PIPE,*
*ERROR_DUCT_INIT_FAILED,*

[10] alpha release only request() gramContact was part of the constructor.
[11] Add example here.

*ERROR_DUCT_LSP_FAILED,*
*ERROR_EXECUTABLE_PERMISSIONS,*
*ERROR_FCNTL_FAILED,*
*ERROR_FORKING_EXECUTABLE,*
*ERROR_GRAM_CONTACT_NOT_SPECIFIED,*
*ERROR_INALID_REQUEST,*
*ERROR_INSUFFICIENT_FUNDS,*
*ERROR_INVALID_COUNT,*
*ERROR_INVALID_HOST_COUNT,*
*ERROR_INVALID_JOBSTATE,*
*ERROR_INVALID_JOBTYPE,*
*ERROR_INVALID_MAXTIME,*
*ERROR_INVALID_MYJOB,*
*ERROR_INVALID_PARADYN,*
*ERROR_INVALID_PROJECT,*
*ERROR_INVALID_QUEUE,*
*ERROR_INVALID_SCRIPT_REPLY,*
*ERROR_INVALID_SCRIPT_STATUS,*
*ERROR_JM_FAILED_ALLOW_ATTACH,*
*ERROR_JOB_CANCEL_FAILED,*
*ERROR_JOB_EXECUTION_FAILED,*
*ERROR_MALLOC_FAILED,*
*ERROR_MPI_NOT_SUPPORTED,*
*ERROR_NO_JOB_ID,*
*ERROR_NO_RESOURCES,*
*ERROR_NULL_SPECIFICATION_TREE,*
*ERROR_OPENING_CACHE,*
*ERROR_OPENING_CACHE_USER_PROXY,*
*ERROR_OPENING_JOBMANAGER_SCRIPT,*
*ERROR_OPENING_STDERR,*
*ERROR_OPENING_STDOUT,*
*ERROR_PARAMETER_NOT_SUPPORTED,*
*ERROR_PROTOCOL_FAILED,*
*ERROR_PROXY_FILE_OPEN_FAILED,*
*ERROR_PROXY_FILE_RELOCATION_FAILED,*
*ERROR_RSL_PARADYN,*
*ERROR_RSL_PROJECT,*
*ERROR_RSL_QUEUE,*
*ERROR_RSL_STDERR,*
*ERROR_RSL_STDIN,*
*ERROR_RSL_STDOUT,*
*ERROR_STDERR_FILENAME_FAILED,*
*ERROR_STDIN_NOTFOUND,*
*ERROR_STDOUT_FILENAME_FAILED,*
*ERROR_SYSTEM_CANCELLED,*
*ERROR_TEMP_SCRIPT_FILE_FAILED,*
*ERROR_UNIMPLEMENTED,*
*ERROR_UNKNOWN_JOB,*
*ERROR_UNSUPPORTED_PARAMETER,*
*ERROR_USER_CANCELLED,*
*GATEKEEPER_MISCONFIGURED,*

*INVALID_JOB_MANAGER_TYPE,*
*RSL_ARGUMENTS,*
*RSL_COUNT,*
*RSL_DIRECTORY,*
*RSL_DRYRUN,*
*RSL_ENVIRONMENT,*
*RSL_EVALUATION_FAILED,*
*RSL_EXECUTABLE,*
*RSL_HOST_COUNT,*
*RSL_JOBTYPE,*
*RSL_MAXTIME,*
*RSL_MYJOB,*
*STAGING_EXECUTABLE,*
*STAGING_STDIN,*
*SUCCESS,*
*VERSION_MISMATCH,*
*ZERO_LENGTH_RSL*

**METHODS:**

**INITIALIZATION**

▷ *GramJobException()*, creates a GramJobException with a SUCCESS value for the error code.

▷ *GramJobException(int errorCode),* creates a GramJobException with the specified error code.

▷ *void setErrorCode(int errorCode*), sets the error code for the exception.

**QUERY**

▷ *int getErrorCode()*, returns the error code contained in the exception.

▷ *String getErrorMessage()*, returns a textual description of an error contained in the exception.

▷ *static String getErrorMessage(int errorCode)*, a static method to return a textual description of an error when given an error code.

### 5.2.4 interface GramJobListener

The interface *GramJobListener* is implemented by objects that wish to receive notification of GramJob state changes.

**INTERFACE NAME** GramJobListener

**METHODS**

▷ *void stateChanged(GramJob job)*, notifies the implementer that the state of the specified GramJob passed has changed.

### 5.3 Package org.globus.duroc

The duroc package allows, oine to use the Globus coallocation mechanism. The package contains the following classes:

- ▷ org.globus.duroc
- ▷ org.globus.duroc.Duroc
- ▷ org.globus.duroc.DurocJob
- ▷ org.globus.duroc.DurocException
- ▷ org.globus.duroc.DurocJobListener

**Example:**

The following example illustrates how easy it is to use Duroc.

```
public class DurocExample implements DurocJobListener {

  private someMethod() {
    ...

    String jobDescription =
                  "&(resourceManagerContact=ico16...)" +
                   "(label=job1)(executable=...)...";
    DurocJob job = new DurocJob();
    job.addListener(this)
    try
      job.addSubJob(jobDescription);
    } catch (DurocException e) {
      // addSubJob failed
      ...
    }

    try {
      job.request();
    } catch (DurocException e) {
      // request failed
      ...
    }

    ...

    // that job isn’t starting, try somewhere else...
    try {
      job.deleteSubJob("job1");
    } catch (DurocException e) {
      // deleteSubJob failed
      ...
    }
    String rsl2 ="&(resourceManagerContact=denali...)"+
                    "(label=job2)(executable=...)...";
    try {
      job.addSubJob(rsl2);
    } catch (DurocException e) {
      // addSubJob failed
      ...
    }

    ...

    // ok, the job is on the nodes
    try {
      job.releaseBarrier();
    } catch (DurocException e) {
      // releaseBarrier failed
      ...
    }

    ...

    // the user wants to do something else so...
    try {
      job.cancel();
    } catch (DurocException e) {
      // cancel failed
      ...
    }
  }

  public stateChanged(DurocJob job) {
    ...
  }

  public stateChanged(DurocJob job, String label) {
    ...
  }
}
```

### 5.3.1 class Duroc

The class *Duroc* is for internal use. No methods should be called by users.

### 5.3.2 Class DurocJob

The class *DurocJob* is the main class for using Globus DUROC to execute applications that run on more than one computer system. This class publicly defines STATE_* variables that represent the possible states of a DurocJob and SUBJOB_STATE_* variables that represent the possible states of subjobs.

**CLASSNAME**

*DurocJob*

**VARIABLES** The following static int variables are defined (their meaning is documented in the Globus Manual):
*STATE_ACTIVE*,
*STATE_CHECKED_IN*,
*STATE_CHECKING_IN*,
*STATE_DONE*,
*STATE_FAILED*,
*STATE_FAILING*,
*STATE_FINISHING*,
*STATE_NOT_SUBMITTED*,
*STATE_PENDING*,
*STATE_STARTING*,
*SUBJOB_STATE_ACTIVE*,
*SUBJOB_STATE_CHECKED_IN*,
*SUBJOB_STATE_DONE*,
*SUBJOB_STATE_FAILED*,
*SUBJOB_STATE_PENDING*,
*SUBJOB_STATE_RELEASED*

**METHODS**

▷ *DurocJob()*, creates a DurocJob.

▷ *void addSubJob(String rsl) throws DurocException*, adds a subjob to the DurocJob. This function must be executed before the releaseBarrier() call is made to the job. The RSL description must contain the GRAM contact contained with the attribute name 'resourceManagerContact'. The RSL description must contain a unique label for the subjob with the attribute name 'label'. A DurocException is thrown if an error occurs. An addSubJob performs a request to a GRAM via the native Duroc interface.

▷ *void deleteSubJob(String label) throws DurocException*, removes a GRAM job from the DurocJob. This operation is valid only if the releaseBarrier method has not been called on the DurocJob. The label argument identifies which subjob to delete. The delete subjobs performs a cancel to a GRAM via the native Duroc call.

▷ *void releaseBarrier throws DurocException*, releases the barrier for the DurocJob. This allows the computation to proceed past the initialization phase and begin computation. If the operation fails, a DurocException is thrown.

▷ *void cancel() throws DurocException*, cancel an already request()ed DurocJob. If the cancel fails.

▷ *String getID()*, returns the ID (Duroc job contact) assigned to the job when it was requested.

▷ *int getState()*, gets the state for the DUROC job. The C DUROC implementation does not use states for the job. The Java code computes the overall state from the subjob state.

▷ *int getState(String label)*, gets the state for the subjob identified by the label.

▷ *String getStateAsString()*, gets the textual representation of the state for the DUROC job.

▷ *String getStateAsString(String label)*, gets the textual representation of the state for the specified subjob.

▷ *String getGramContact(String label)*, returns the contact for the Gram that the job with the specified label will be or has been submitted to.

▷ *String getDescription(String label)*, returns the RSL description of the subjob identified by the label.

▷ void *addListener(DurocJobListener listener)*, adds a listener to this DurocJob. All listeners are notified when the state of a DurocJob changes.

- ▷ void *removeListener(DurocJobListener listener)*, removes a listener from this DurocJob. The listener will no longer be notified when the state of the DurocJob changes.

### 5.3.3 Class DurocException

The class *DurocException* contains many definitions for error codes of the form ERROR_* and also SUCCESS. The error codes are a superset of those defined for the Duroc C client. Additional error codes are added for errors that can occur in the Java code.

**CLASSNAME**

*DurocJobException*

**VARIABLES** The following variables are defined (their meaning can be found in the Globus manual):
*ERROR_ALREADY_CANCELLED,*
*ERROR_ALREADY_RELEASED,*
*ERROR_BAD_COMMS_TYPE,*
*ERROR_BAD_START_TYPE,*
*ERROR_DUCT_FAILED,*
*ERROR_DUPLICATE_SUBJOB_LABEL,*
*ERROR_DUROC_ALREADY_EXISTS,*
*ERROR_GRAM_CONTACT_NOT_SPECIFIED,*
*ERROR_GRAM_FAILED,*
*ERROR_INIT_FAILED,*
*ERROR_INTERNAL_FAILURE,*
*ERROR_INVALID_CHECKIN,*
*ERROR_INVALID_MANAGER_CONTACT,*
*ERROR_INVALID_MULTIREQ,*
*ERROR_INVALID_OPERATION,*
*ERROR_INVALID_PARAMETER,*
*ERROR_INVALID_REQUEST,*
*ERROR_INVALID_RSL,*
*ERROR_LABEL_NOT_SPECIFIED,*
*ERROR_MALLOC_FAILED,*
*ERROR_NEXUS_FAILED,*
*ERROR_NOT_INITIALIZED,*
*ERROR_PROTOCOL_VERSION_MISMATCH,*
*ERROR_UNKNOWN_JOB,*
*ERROR_UNKNOWN_LABEL,*
*SUCCESS*

**METHODS**

- ▷ *DurocJobException()*, create a DurocJobException with a SUCCESS value for the error code.
- ▷ *DurocJobException(int errorCode)*, create a DurocJobException with the specified error code.
- ▷ *void setErrorCode(int errorCode)*, set the error code for the exception.
- ▷ *int getErrorCode()*, returns the error code contained in the exception.
- ▷ *String getErrorMessage()*, returns a textual description of the error contained in the exception.
- ▷ *static String getErrorMessage(int errorCode)*, returns a textual description of an error when given an error code.

### 5.3.4 Interface DurocJobListener

The interface *DurocJobListener* is an interface implemented by objects that wish to receive notifications of DurocJob state changes and subjob state changes.

**INTERFACE NAME**

*DurocJobListener*

**METHODS**

▷ *void stateChanged(DurocJob job)*, notifies the implementer that the state of the specified DurocJob has changed.

▷ *void stateChanged(DurocJob job, String label)*, notifies the implementer that the state of a subjob identified by the label of the specified DurocJob has changed.

## 5.4 Package org.globus.mds

### 5.4.1 Class MDS

The intention of the MDS class is to simplify the connection to the Metacomputing Directory Service (MDS). The MDS class achieves this by

▷ establishing a connection to an MDS Server,

▷ querring for contents in the MDS,

▷ printing the result of a query, and

▷ disconnecting from the MDS Server.

The main motivation for this class is to have an intermediate application layer that can be easily adapted to different LDAP client libraries (JNDI, Netscape SDK, Microsoft SDK). To adapt to the different ports only a limited number of routines have to be rewritten. The current release is based on JNDI.

**Connecting.** The first step in using the class is to establish a connection, which is done in the following way:

```
MDS mds = new MDS("www.globus.org");
```

The parameters to the MDS class constructor are simply the DNS name of the MDS server and the port number for the connection. The default values for the connection to the MDS are set to *ldap://mds.globus.org:389*. These values are used if no parameters are specified:[12]

```
MDS mds = new MDS();
```

**Searching.** Searching the MDS is done with LDAP queries with respect to a base directory specified by a distingished name (DN). The top level of a Globus related searches is

```
o=Globus, c=US
```

A search is invoked in the following way:

```
MDSResult result = mds.search("o=Globus, c=US", "(object-
class=*)", MDS.ONELEVEL_SCOPE);
```

The search result is stored in a MDSResult data structure.

[12]In future this might be changed to be the bas DN of the organization.

**Printing.** A result from a search can be modified by the appropiate JNDI functions. A simple print function is provided that is intended to simplify debugging and programming development:

```
mds.print(result, false);
```

**Example.** The following example shows a program which connects to the MDS, and prints objects of type *Globus-Person* located at Argonne National Laboratory:

```
MDS mds = new MDS("mds.globus.org", "389");
try {
    mds.connect();
    String bindDN = "o=Argonne National Laboratory, " +
                                "o=Globus, c=US";

    MDSresult result = mds.search(bindDN,
                      "(objectclass=GlobusPerson)",
                       MDS.SUBTREE_SCOPE);
    System.out.println(results);
    mds.disconnect();
}catch(MDSException e) {
    System.err.println( "Error:"+ e.getLdapMessage() );
} finally {
    mds.disconnect();
}
```

**VARIABLES:**

| Name | Description |
|---|---|
| *OBJECT_SCOPE* | see Globus Manual |
| *ONELEVEL_SCOPE* | - ” - |
| *SUBTREE_SCOPE* | - ” - |

**METHODS:** Unless otherwise noted, the parameters to search functions have the following meaning:

| | |
|---|---|
| *base* | the base DN from which a search is issued |
| *filter* | a common LDAP filter |
| *searchScope* | sets the scope of the search |
| *attributesToReturn* | specifies the attributes to be returned |

Furthermore, an *MDSExecption* is thrown if an error occurs.

**Initializing**

▷ *MDS()*, initializes a connection to the MDS database with its default values

▷ *MDS(String hostname, String port)*, initilaizes a connection to the MDS database to the specified host and port
Parameters:
*hostname* - specifies the hostname to connect to
*port* - the port on which the conection is established

**Server Related Methods**

▷ *void setPort(String port)*, sets the port to the specified value.

▷ *String getPort()*, gets the port

▷ *void setHostname(String hostname)*, sets the hostname to the specified value

▷ *String getHostname()*, returns the hostname

▷ *void setSearchTimeout(int newTimeout)*, sets the search timeout in seconds.

▷ *int getSearchTimeout()*, returns the search timeout in seconds

▷ *void setSearchLimit(int newLimit)*, sets the search limit - number of entries that will be returned

▷ *int getSearchLimit()*, returns the search limit

▷ *void setLdapVersion(int newVersion)*, sets LDAP protocol version (allowed values are 2 and 3).

▷ *int getLdapVersion()*, returns the LDAP protocol version

▷ *String getURL()*, returns the URL

**Connecting**

▷ *void connect() throws MDSException*, connects to the specified server. An *MDSException* is thrown if the conection to the server can not be established using the default values (hostname=mds.globus.org and port=389).

▷ *void connect(String managerDN, String password) throws MDSException*, connects and authenticates to the specified server. The parameters are as follows:
managerDN - specifies the distingushed name of the directory manager or user
password - specifies the password of the directory manager or user

▷ *void disconnect() throws MDSException*, disconnects from the MDS server and throws an MDSException on failure to disconnect.

▷ *void reconnect() throws MDSException,* reconnects to the server and throws an MDSexception on failure to reconnect.

**Search**

▷ *MDSResult search(String baseDN, String filter, int searchScope) throws MDSException*, issues a specified search to the MDS starting from the base DN. The search returns all attributes.

▷ *MDSResult search(String baseDN, String filter, String[] attrToReturn, int searchScope) throws MDSException*, issues a specified search to the MDS starting from the base DN.

▷ *MDSResult list(String baseDN) throws MDSException*, returns the objects bound to the DN in a Hashtable.

**Attribute Querry**

▷ *Vector getAttributeNames(String dn) throws MDSException*, returns the names of the attributes specified by the dn in a vector in order of their occurrence.

**Deleting**

▷ *void deleteEntry(String dn) throws MDSException*, deletes a specific object specified by the dn.

▷ *void deleteTree(String dn, boolean deleteItSelf) throws MDSException*, deletes a subtree starting from the specified DN. If deleteItSelf is set to true, it also deletes the object identified by the DN.

▷ *void deleteAttribute(String dn, String attribute) throws MDSException*, deletes the attribute and all its values in the object specified by dn.

▷ *void deleteValues(String dn, MVHashtable attributes) throws MDSException*, deletes the attributes identified by the MVHashtable.

**Renaming**

▷ *void renameEntry(String oldDN, String newDN, boolean deleteOldDN) throws MDSException*, renames an object in an LDAP database. When renaming an entry, one has the option ofnot keeping the entry's old relative distinguished name as an attribute of the updated entry. This is activated by setting deleteOldDN to true (see the JNDI manual).

**Updating**

▷ *void addEntry(String dn, MDSAttribute attributes) throws MDSException*, adds the specified attributes to the object identified by dn.

▷ *void updateEntry(String dn, MDSAttribute attributes) throws MDSException*, updates (adds or replaces) attributes of an entry.

▷ *void addAttribute(String dn, MDSAttribute attributs) throws MDSException*, adds attributes to an entry.

### 5.4.2 Class MDSResult

**CLASSNAME:**

*MDSResult*

**PURPOSE:**

The class MDSResults extends Hashtable and holds a set of MDSAttibutes that are returned by MDS queries (see Section 5.1.1). The key to each MDSAttribute is the corresponding DN.

**METHODS:**

**Initialization**

▷ *MDSResult()*

**Other Methods**

▷ the same methods as defined in the class java.util.Hashtable.

▷ *void print()*, prints the data structure in readable format.

#### 5.4.3 Class MDSAttribute

**CLASSNAME:**

*MDSAttribute*

**PURPOSE:**

The class MDSAttibute extends MVHashtable to function as a data structure that stores results returned in by MDS queries (see Section 5.1.1). It stores attributes that are returened by many MDS related methods.

**METHODS:**

The methods are the same as defined in MVHashtable.

### 5.5 Package org.globus.gass

The purpose of the *gass* package is to

▷ simplify the file transfers,

▷ provide a graphical browser of remote files at several sites with drag and drop copying, moving, and deleting of files,

▷ introduce the concept of GassException,

▷ enable the monitoring of exceptions with the help of listeners (GassClientListener and GassServerListener).

More will be here shortly.

### 5.6 Package org.globus.gara

More will be here shortly.

### 5.7 Package org.globus.hbm

More will be here shortly.

# Part II
# High-Level Packages

More will be here shortly.

# Part III
# Graphical User Interface Components

More will be here shortly.